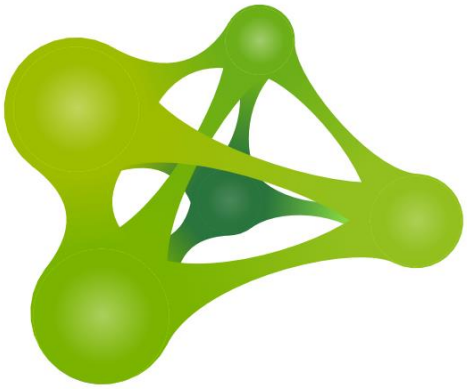
NE1ONE

UNDERSTAND | PREDICT | MANAGE

EMULATOR

Command Line Interface (CLI) Scripting & API Guide



Version 3.0 March 2017

Global Support Email: support@itrinegy.com

Regional Telephone Hotline Support:

Americas: 1-888-448-4366 EMEA: +44 (0)1799 252 200

Contents

1 Purpose and Applicability

The purpose of this guide is to:

- Explain how to control the Emulator from the command line or a TCP sockets interface

2 Introduction

The Emulator's GUI and LCD Panel send commands to setup and control the Emulator.

The command line and sockets API interface provide the ability for you to perform important functions within your own scripting or programming.

This document covers all CLI and API options available in the Emulator which at this time are:

- Start and stop an emulation
- List all the defined emulations
- List all the defined emulations filtered by types:
    - Point to Point (ptp)
    - Multi Hop
    - Profiled
- Show the running emulations
- Stop all the running emulations

3 Emulator Concepts

It is worth being familiar with the concepts and operation of the Emulator as explained in the *NE-ONE - Emulator User & Administrator Guide* before reading this guide or attempting to use the command line or API.

Here are the essentials when it is boiled down:

The Emulator transfers packets from one physical network port (or VM equivalent, when it's running in a VM environment) to another physical network port. Whilst transferring the packet the Emulator can perform certain actions on the packet, such as delaying the packet, losing the packet and damaging the packet.

There are three major concepts in the Emulator:

1. Physical Ports and Port Pairs
2. Emulations
3. Links

These are detailed in the sections below.

3.1 Physical Ports and Port Pairs

Physical ports connect the Emulator to the external network. The ports are used in port pairs:

- Port Pair 0 & 1
- Port Pair 2 & 3

The latter port pair is not available on all models.

Essentially packets entering one port in a port pair will emerge from the other port, unless the packet is (deliberately) lost.

3.2 Emulations

Emulations consist of one or more Links (see next section), with appropriate settings for each link, which have been named and saved on the Emulator.

There may be many emulations stored on the Emulator, but only one emulation can be started, on a Port Pair, at a time.

Emulations are separated into three types:

- Point to Point
- Multi Hop
- Profiler

The User guide describes these fully, but the API will divide emulations into these categories when asked for a list.

3.3 Links

Links allow multiple network circuits to be provided simultaneously by channelling appropriate traffic (by IP address, IP Port, VLAN id) through a particular link.

Which link traffic will use is determined by Link Qualification Criteria setup when the emulation is defined in the GUI.

The process is that packets enter a port in a port pair (for example port 0) and then based on link qualification criteria are directed to a link. The link then applies the defined network limitations (impairments) and the packets are recombined from the separate links and output through the other port in the port pair (port 1 in our example).

The main user guide describes this in detail.

4 Command Line Requirements

The Emulator can be scripted in one of 2 ways:

1. By using the supplied sample Python command line script called ***necli.py*** (clearly this will need an installed python interpreter)
2. By creating an application that creates a TCP socket and connects directly to the Emulator using TCP port 7292 using SSL

When using the command line script or API there are certain requirements that must be met in order to get a successful outcome. Whether used from the API or from the CLI, all of the command options are prefixed with '--' (two dashes) to help them stand out, and, where there is an argument the argument should be quoted to avoid issues with spaces, also a valid username and password must be supplied for the Emulator. e.g. from Windows (assuming the necli.py is available in the local folder and a python interpreter has been installed as is in the PATH) you can issue commands like this:

```
C:\Users\Lara> python necli.py --host 192.168.202.194 --user admin --password admin --getemulationsbyportpair
```

Some of the commands require more than one parameter, where that is the case then there is still only one argument but the parameters are separated by a semi-colon.

The necli.py script sends all of the commands (except for --host) directly to the Emulator and echo's any responses direct to the terminal (stdout). Responses from the Emulator are also in the form of commands and values. For commands that do not have a specific response then the command "`--ok`" is returned. If there is an error when executing the command then it will return `--error "<Message>"`. Below are some examples along with their return values:

```
C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --getlicense ports

→ --license "ports;4;"

C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --notACommand

→ --error "Option [--notACommand] Unknown"
```

The examples below all assume the command line python script necli.py is being used. For full information on how to use the command line and also the sockets API please see section 8 Methods of Issuing Emulator Commands on page 28.

5 Starting, Stopping and Listing Standard Emulations

This section documents all the commands related to locating defined standard emulations, as well as starting & stopping standard emulations. Standard Emulations refers to Emulations that are not Scenarios (i.e. not changing between standard emulations over time)

5.1 Listing All Defined Standard Emulations

Standard Emulations are defined (created and saved) by the web GUI. They can be launched by the LCD panel and CLI/API, but the question is what standard emulations are available. This command lists all the saved standard emulations present on the Emulator.

Syntax is as follows:

```
--getDefinedEmulations [--type (ptp|multihop|profiled)]
```

This lists all the current defined standard emulations belonging to the *--user {user}* or public stored on the Emulator. If supplied with optional filter argument *--type*, then it only returns all the defined standard emulations for that particular type.

Output:

```
--names "{number of results};{type 1};{name 1};{type 2};{name 2};{type 3};{name 3};..."
```

Here, {number of results} is the number of defined standard emulations found on the Emulator for the user (this includes public emulations).

Example:

```
$ python necli.py --host 192.168.202.194 --user admin --password admin --getdefinedemulations
```

→ *--names 12;ptp;3G_Slow_PoorQuality;ptp;Satellite_Slow_PoorQuality;multihop;Wan10MbpsToWan155Mbps;ptp;WAN_10Mbps_PoorQuality;ptp;2G_Slow_GoodQuality;multihop;WiFi56MbpsToADSLGood;ptp;ADSL_Medium_GoodQuality;ptp;LAN_No_Impairment;ptp;WiFi_56Mbps_PoorQuality;ptp;LAN_1Gbps_AvgQuality;ptp;4G_Slow_PoorQuality;ptp;SDSL_Slow_GoodQuality;*

This says there are 12 standard emulations and then list them as type;name pairs

5.2 Starting a Standard Emulation

To start any standard emulation, it must be present on the Emulator. Standard Emulations are started by specifying their name and the port pair the standard emulation is to be started on:

Syntax is as follows:

```
--startEmulation {name} --portPair (0|1)
```

This starts a standard emulation with the *name* specified (if in doubt you can get a list of defined standards emulations from *--getDefinedEmulations*) on the specified port pair.

Output:

--ok if the standard emulation name is successfully started on port pair. Otherwise *--error <error message>* if there is already an emulation on the port pair or emulation *name* doesn't exist. <error message> will indicate the actual error

Example:

```
$ python necli.py --host 192.168.202.194 --user admin --password admin --startEmulation 3G_Slow_PoorQuality --portPair 0
```

→ `--ok`

And if you try again without stopping the first:

```
$ python necli.py --host 192.168.202.194 --user admin --password admin --startEmulation 3G_Slow_PoorQuality --portPair 0
```

→ `--error "Emulation is already running on port pair 0"`

Note

If the emulation name contains a space then the name will need to be quoted e.g. *–startEmulation "Mobile Test 1"*

5.3 Stopping a Standard Emulation

This stops whatever emulation is running on the specified port pair. After stopping the emulation no traffic will be passed on on that port pair.

Syntax is as follows:

```
--stopEmulation --portPair (0|1)
```

If no emulation is running on that port pair it will return an error.

Output:

--ok if a running emulation is stopped successfully. If there is no emulation running on the port pair, the command will output *--error "<error message>"*

Example:

```
C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --stopemulation --portPair 0
```

→ *--ok*

And if you try it again, now that there is no emulation running:

```
C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --stopemulation --portPair 0
```

→ *--error "No emulation is running on the port pair"*

5.4 Stop all Standard Emulations

This will stop all emulations running on any port pair. It's a good way of ensuring you're back to the Emulator's initial state with no emulations running.

Syntax is as follows:

--stopAllEmulations

Unlike `--stopEmulation`, this command will always return `--ok` even there is no emulation running on any port pair.

Output:

--ok in all cases

Example:

C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --stopallemulations

→ *--ok*

5.5 Show the Running Emulations

This will show you what emulations are running, either in total for all port pairs available, or for a specified port pair.

Syntax is as follows:

--getEmulationsByPortPair [--portPair (0|1)]

Lists the currently running emulations for both port pairs 0 (ports 0 and 1) and 1 (ports 2 and 3). If supplied with the optional *--portPair <number>* parameters, then the command only returns the emulations running on the port pair specified by <number>.

Output:

Without the *-portPair <number>* argument:

--emulations "{number of results};{port pair id};{emulation name};{port pair id};{ emulation name};"

With --portPair <number> arguments:

--emulations "1;{port pair id};{name}"

Example:

```
C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --getEmulationsByPortPair
```

→ *--emulations "2;0;3G_Slow_PoorQuality;1;;"*

So its saying – 2 emulations follow… First, port pair 0 is running 3G_Slow_PoorQuality, then, port pair 1 is running nothing (;;)

5.6 List Compatible (Similar) Standard Emulations with a Started Emulation

This will produce a list of all the **Compatible** standard emulations started on a portpair.

For an emulation to be ***Compatible*** with the current one it must have the same links in the same positions. The links and end-points do not need to have the same names or be in the same states (enabled/disabled). Implicitly, therefore, multihop emulations are not compatible with single hop emulations.

Syntax is as follows:

```
--getCompatibleEmulations --startedOnPortPair (0|1)
```

Output:

```
--names "{number of results};{type};{emulation name};{type};{emulation name};"
```

Example:

```
C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --getCompatibleEmulations --startedOnPortPair 0
```

→ *--names "{9;ptp;2G_Slow_GoodQuality;ptp;4G_Slow_PoorQuality;ptp;ADSL_Medium_GoodQuality;ptp;LAN_1Gbps_AvgQuality;ptp;LAN_No_Impairment;ptp;SDSL_Slow_GoodQuality;ptp;Satellite_Slow_PoorQuality;ptp;WAN_10Mbps_PoorQuality;ptp;WiFi_56Mbps_PoorQuality;"*

5.7 List Compatible (Similar) Standard Emulations with a Named Emulation

This will produce a list of all the **Compatible** standard emulations to named (template) standard emulation. It operates in a very similar way to section 5.6: List Compatible (Similar) Standard Emulations with a Started Emulation, above. See that section for the definition of a compatible emulation.

Syntax is as follows:

--getCompatibleEmulations --templateEmulation <name>

Where <name> is the name of a "template" standard emulation i.e. the one which is used as a basis for determining compatibility.

Output:

--names "{number of results};{type};{emulation name};{type};{emulation name};

5.8 Dynamically change to Emulation without Stopping

This allows you to change (update) the current running standard emulation parameters (on a port pair) to those of a Compatible (Similar) standard emulation.

Links will retain the names of the emulation that was started with –start or via the GUI). It does not matter if links are enabled or disabled differently, their status will change as required.

Syntax is as follows:

--changeToEmulation <name> --portPair (0|1) [--force]

The option *--force* will perform an emulation stop, and then start if the new emulation is not compatible with the running standard emulation.

Examples:

1) Compatible change
   C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --changeToEmulation 2G_Slow_GoodQuality -portPair 0
   → *--ok*

2) Not compatible change, no force
   C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --changeToEmulation Wan10MbpsToWan155Mbps --portPair 0
   → *--error "Cannot change to emulation: incompatible configuration"*

3) Not compatible change, force
   C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --changeToEmulation Wan10MbpsToWan155Mbps --portPair 0 --force

→ *--ok*

6 Playing, Stopping and Listing Scenarios

These commands relate to the Scenario functionality in the NE-ONE i.e. the ability to play time based emulations. These are created by putting together one or more (usually more) standard emulations or scenarios (i.e. scenarios can include other scenarios too) with transitions with the NE-ONE GUI's Scenario editor. It will be worth familiarising yourself with the operation of Scenarios in the GUI before using these API commands

At this time the API concerns itself solely with playing (running) such timed Scenarios not creating or editing them which is done in the scenario editor GUI

6.1 Get a List of Scenarios

This allows you to get a list of all the available scenarios, their types (i.e. point to point or multi-hop), durations and end mode (i.e. at the end of the scenario: Stop the emulation, Stay in the last configuration, Go back to the beginning).

Syntax is as follows:

--listScenario

Output:

--scenarios "{number of scenarios};{name of scenario};{emulation type};{duration};{end mode};...."

Where:

number of scenarios - is number of defined scenarios that are VALID (i.e. ready to play).

Each valid scenario is then listed and comes with 4 fields:

- Name of Scenario - Scenario name
- Emulation type – the emulation type, 'ptp' or 'multihop'
- Duration – the total duration of the Scenario (in seconds)
- End mode – the Scenario's behaviour when reaching the end (1 - Stop Emulation, 2 - Stays On with the parameters from the last element of the scenario, 3 – Repeat i.e. go back to the beginning)

Examples:

C:\Users\Lara>python necli.py --host 192.168.202.135 --user admin --password admin --listScenario

→ --scenarios
"3;Dual_Link_1;ptp;60;1;Dual_Link_2;ptp;60;1;Single_Link_1;ptp;100;1;"

6.2 Play (Run) a Scenario

This allows you to run a scenario by name specifying the point pair on which it should run and the playback speed (the default is 1x – normal speed)

Syntax is as follows:

--playScenario {Scenario Name} --portPair {Port Pair} [--speed {Speed} (one of 1|2|4|0.5|0.25)]

Output:

--ok (if successful) or

--error "{reason}" (if there was a problem – the reason specifies the issue)

Where:

- *Scenario Name* - the scenario name to play
- *Port Pair* - which Port pair to start an scenario on e.g. 0 or 1
- *Speed* – is the speed to run the scenario at. Valid values are 1, 2, 4, 0.5, 0.25 which will play the scenario faster (2, 4), at normal speed (1) or slower (0.5, 0.25). The speed value is a divisor for how long 1 second of scenario will actually take. If speed is not specified, it will be 1 which is normal speed (1x)

Examples:

C:\Users\Lara>python necli.py --host 192.168.202.135 --user admin --password admin --playScenario Dual_Link_1 --portpair 0

→ --ok

The scenario now plays (in the background).

Note

If the scenario name contains a space then the name will need to be quoted e.g. --playScenario "My Scenario"

6.3 Pause a Playing (Running) Scenario

This will pause the scenario running on a port pair i.e. it will freeze the timer but leave the emulation in the state that it is currently in (i.e. the emulation will be running).

Syntax is as follows:

```
--pauseScenario --portPair {portPair}
```

Output:

```
--ok
--error "{reason}"
```

Where:

{portPair} is either 0 for the first port pair (ports 0 & 1) or 1 for the second port pair (ports 2 & 3)

Examples:

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --pauseScenario --portpair 0

→ --ok

6.4 Resume a Paused Scenario

This will resume a previously paused scenario running on a port pair i.e. it will resume the timer from where it was paused and the scenario will continue to run from that point

Syntax is as follows:

```
--resumeScenario --portPair {portPair}
```

Output:

```
--ok
```

--error "{reason}"

Where:

{portPair} is either 0 for the first port pair (ports 0 & 1) or 1 for the second port pair (ports 2 & 3)

Examples:

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --resumeScenario --portpair 0

→ --ok

6.5 Stop a Playing (Running) Scenario

This will stop the scenario running on a port pair i.e. the timer will be stopped and the emulation will be stopped.

Syntax is as follows:

--stopScenario --portPair {portPair}

Output:

--ok

--error "{reason}"

Where:

{portPair} is either 0 for the first port pair (ports 0 & 1) or 1 for the second port pair (ports 2 & 3)

Examples:

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --stopScenario --portpair 0

→ --ok

6.6 Set Scenario Speed

This has two functions:

1. To set the default playback speed for a port pair

2. To change the speed at which a currently running scenario is being played back – this value will then be remembered for the next time a scenario is started on the port pair

Syntax is as follows:

```
--setSpeed {Speed} (one of 1|2|4|0.5|0.25) --portPair {portPair}
```

Output:

```
--ok
--error "{reason}"
```

Where:

{Speed} is one of 1|2|4|0.5|0.25. They will play the scenario faster (2, 4), at normal speed (1) or slower (0.5, 0.25). The speed value is a divisor for how long 1 second of scenario will actually take.

{portPair} is either 0 for the first port pair (ports 0 & 1) or 1 for the second port pair (ports 2 & 3)

Examples:

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --setSpeed 0.25 --portPair 0

→ --ok

This example runs the scenario at ¼ of normal speed i.e. the scenario and each element in it takes 4 times as long to complete. If the scenario is running in the GUI the scenario progess bar moves at ¼ of its normal rate

NOTE

Issuing the –setSpeed CLI or API does not immediately change the value in the GUI's dropdown speed menu (if the GUI is being used). However, if you play a scenario in the GUI it will accept the change of speed and the GUIs dropdown changes at that moment.

6.7 Advance to Next Scenario Frame

This will advance the scenario to the next Scenario Frame i.e. to the start of the next Element, which may be a transition or an Emulation. If it currently running a transition when the command is received, it will abandon it and set the values for the next Emulation Element.

Syntax is as follows:

--nextScenFrame --portPair {portPair}

Output:

--ok

--error "{reason}"

Where:

{portPair} is either 0 for the first port pair (ports 0 & 1) or 1 for the second port pair (ports 2 & 3)

Examples:

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --nextScenFrame --portPair 0

→ --ok

NOTE

If you are in the last element of the scenario it will not advance to the end, instead giving an error saying "Cannot fast forward: already at the end of the scenario"

6.8 Go Back to Previous Scenario Frame

This will take the Scenario back to the start of the previous Scenario Frame i.e. to the start of the previous Element, which may be a transition or an Emulation. If it currently running a transition when the command is received, it will abandon it and set the values for the previous Emulation Element.

Syntax is as follows:

--prevScenFrame --portPair {portPair}

Output:

--ok

--error "{reason}"

Where:

{portPair} is either 0 for the first port pair (ports 0 & 1) or 1 for the second port pair (ports 2 & 3)

Examples:

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --prevScenFrame --portPair 0

→ --ok

NOTES

- Unlike a Music player it *will not* take you back to the start of the current element if more than a few seconds of the element have passed. To achieve this use –prevScenFrame and immediately afterwards --nextScenFrame
- If you are in the first element of the scenario it will not advance to the beginning, instead giving an error saying "Cannot rewind: already at the start of the scenario"

6.9 Get Scenario Running Information

This gets information on Scenarios running by portpair. You can see if scenarios are running and, if running, the name of the Scenario.

Syntax is as follows:

--getScenarioByPortPair [--portPair {portPair}]

Output:

With the optional portPair argument:

--status "1;{portPair};{running};{scenario name};"

Without the optional portPair argument:

--status "{Number of ports};0;{running for portpair 0};{name of scenario for portpair 0};…" (for each portpair)

Where:

- *{portPair}* is either 0 for the first port pair (ports 0 & 1) or 1 for the second port pair (ports 2 & 3)
- *{running}* is 0 if a Scenario is running on that portpair
- *{scenario name}* is the name of the scenario if there is one running on that portpair

Examples:

Here there are two port pairs (4 ports) and no scenarios are running on any portpair:

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --getScenarioByPortPair

→ --status "2;0;0;;1;0;;"

Here there are two port pairs (4 ports) and a Scenario called "Dual_Link_1" is running on portpair 0:

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --getScenarioByPortPair

→ --status "2;0;1;Dual_Link_1;1;0;;"

In the next two examples there are two port pairs (4 ports) and a Scenario called "Dual_Link_1" is running on portpair 0:

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --getScenarioByPortPair --portpair 0

→ --status "1;0;1;Dual_Link_1;"

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --getScenarioByPortPair --portpair 1

→ --status "1;1;0"

6.10 Get Running Scenario Details

This gets details on Scenarios running by portpair. You can see the status of scenarios, when they were started, the Element name, remaining time, speed they are running at, and error status.

Syntax is as follows:

```
--scenarioStatus [--portPair {portPair}]
```

Output:

With the optional portPair argument it returns details only for the portpair specified, without the optional portPair argument it returns the number of scenarios and details for each portPair:

--status "{number of portpairs};{portPair};{scenario name};{status};{at time};{started At};{element name};{time left};{speed};{error}…"

--error "Error message"

Where:

- *{portPair}* - is either 0 for the first port pair (ports 0 & 1) or 1 for the second port pair (ports 2 & 3)
- *{scenario name}* - is the name of the scenario if there is one running on that portpair
- *{status}* - current status of the running scenario. Possible values are STOP, PLAY, PAUSE indicating the status of the scenario is currently playing
- {at time} – the time from the start scenario in seconds
- {started At} - UTC time (in milliseconds) when the scenario is started (Unix time in ms)
- {element name} – the name of the Emulation Element currently in force – if it is a transition the name will be Gradual, Variable or Outage
- {time left} – the time left in the current element, named above
- {speed} – the speed the Scenario is running at. It will be on of 1|2|4|0.5|0.25. The speed value is a divisor for how long 1 second of scenario will actually take.
- {error} – indicates any error so far during the running scenario (1 - error, 0 - no error)

Examples:

In the next two examples there are two port pairs (4 ports) and a Scenario called "Dual_Link_1" is running on portpair 0:

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --scenarioStatus --portpair 0

→ --status "1;0;Dual_Link_1;PLAY;11;1490352881053;Test_twolink_1;9;1;0"

*So, on Portpair **0** a scenario called **Dual_Link_1** is running. It's status is **PLAY**, it has been running for **11** seconds, the absolute start time of the scenario in UTC ms time is **1490352881053**, the current element is **Test_twolink_1** with **9** seconds left, for that element, running at speed **1** (1x = Normal) and Error = **0** means no current error.*

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --scenarioStatus --portpair 1

→ --error """Scenario is not loaded or running on the port pair"\n"'

In the next example a Scenario called "Dual_Link_1", running on portpair 0 is in the Element "Gradual" i.e. a Gradual transition:

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --scenarioStatus --portpair 0

--status "1;0;Dual_Link_1;PLAY;27;1490354519057;Gradual;12;1;0"

*You can see the word **Gradual** in the element position and there is **12** seconds of Gradual left to run.*

6.11 Get Scenario Errors

This will retrieve a list of errors from the currently running scenario.

Syntax is as follows:

```
--getScenarioErrors --portPair {portPair} [--reset]
```

Output:

```
--errors "{time};{message};{time};{message}..."
```

Where:

- *--portPair {portPair}* –is a switch to choose which portpair. either 0 for the first port pair (ports 0 & 1) or 1 for the second port pair (ports 2 & 3)
- *--reset* - is an optional command switch which will reset the error list after returning them
- *{time}* - at which second the error occurred relataive to the Scenario start
- *{message}* - details of the error

Examples:

C:\Users\Lara>python necli.py --host 192.168.202.135 --user admin --password admin --getScenarioErrors –portPair 0

(if no errors) → --errors

(if no scenario running) → --error "Scenario is not loaded or running on the port pair"

6.12 Run scenario Logging (advanced)

These commands are used for the automated testing of the CLI/API should only be used under the guidance of your Support representative, not least due to very high amounts of output produced which can affect the performance of the scenario server.

6.12.1 --setScenarioRunLogs

Syntax is as follows:

--setScenarioRunLogs [--portPair {portPair}] [--runLogDir {dir}]

This will turn on the run log feature in the scenario server. If portPair argument is not given, then it will switch on run time logging for all the port pairs. If runLogDir is not given, then the default directory for creating run time log files is /tmp.

- The log filename is the same as the scenario name with '.log' extension instead and port pair suffix. For example: Testing.scen is loaded and run in portpair 0, the log file is Testing_0.log. If the log file is existed from previous run, it will be renamed to Testing_0.log.old
- The log file ONLY contains entries during the lifetime of a running scenario.
- Each entries is a single line with newline terminated representing a single interval.
- Each interval line is prefixed with "Interval {secs}:"
- Not all interval entries are logged into the scenario log file. Intervals with notable action are logged, such as
- Start/End of the scenario
- Start/End of a component
- Pause/Resume of scenario
- Speed change
- Component skip
- (TBD) micro intervals of transition

6.12.2 --setScenarioRunLogs

Syntax is as follows:

--setScenarioRunLogs [--portPair {portPair}] --off

This will turn off the run log feature:

6.12.3 --getScenarioRunLog

Syntax is as follows:

--getScenarioRunLog {scenario name} --portPair {portPair} [--interval {interval}]

This will return the log entries of the scenario previously run. If --interval is not given, then all log entries are returned.

Where:

--interval accepts the following forms:

- -{num} -- means logs from interval 0 to interval num
- +{num} -- means logs from interval num+1 to the latest logs
- {num1}-{num2} -- means logs between the interval
- {num1},{num2},{num3} -- means a list of specific logs at specific intervals

6.12.4 --playScenario with –logtrans

Syntax is as follows:

--playScenario {Scenario name} --portpair {portPair} --logtrans

This starts logging which includes the transition steps.

Example:

C:\Users\Lara> python necli.py --host 192.168.202.135 --user admin --password admin --playScenario Dual_Link_1 --portpair 0 --logtrans

--ok

7 Additional Commands

There are several commands that are not directly related to emulations that help to provide information on the emulation and the general configuration of the Emulator.

7.1 Reboot the Emulator

This reboots the Emulator

Syntax is as follows:

```
--reboot
```

The Emulator responds with `--ok` and then reboots.

Output:

```
--ok
```

Example:

```
C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --reboot
```

→ `--ok`

7.2 Get License Details

Details of a part of the product license can be retrieved by specifying one of the identifiers in the table below.

Syntax is as follows:

--getLicense <Identifier>

The <Identifier> indicates what license information is required. The table below shows the list of <Identifiers> that can be requested:

Identifier	Description
Ports	The list of ports that are licensed. A comma separated list of ports is returned
Links	The number of licensed links
Bandwidth	The maximum bandwidth allowed
ProductId	The Product Identifier Number (18 for Model 1, 19 for Model 5, 20 for Model 10 and 21 for Model 20)

Output:

--license "<Identifier>;<items>…"

Example:

The example shows the retrieval of the ProductId:

C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --getLicense "ProductId"
→ *--license "ProductId;14"*

7.3 Get License Status

This get the status – licenced or not, and if a temporary licence exists the license expiry date.

Syntax is as follows:

--getLicenseStatus

Output:

--licenseStatus <N>;<expiry date>|permanent

Where

<N> = 1 if the licence is valid and current, and 0 if it does not exist or has expired.

<expiry date> in the form YYYY-MM-DD is the date the licence expires or the word **permanent** if there is no expiry.

Example:

C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --getLicenseStatus

→ *--licenseStatus 1;2015-12-20*

7.4 Get Ports

This command returns the Ports in the Emulator, whether they are available or in use by an emulation.

Syntax is as follows:

--getPorts

Output

--ports "<number of ports>;<port id1>;<name1>;<port parent id1>;<emulation_id1>..."

where the numbers of ports returned is <number of ports> and then there are 4 values per port:

<port idN> - the internal ID of the port

<nameN> - the name of the port

<port parent idN> - the id of the parent port of this port - if any (-1 if no parent)

<emulation_idN> - the id of the emulation which this port is assigned (-1 if there is no emulation)

If the ports have not been assigned IP addresses then they will have the names 0,1,2,3, if they have been assigned an address then you will see names like Port0, Port1. In the latter case the port parent id will tell you what the parent port is.

Example

C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --getPorts

→ *--ports "4;2;2;-1;-1;3;3;-1;-1;4;Port0;0;0;5;Port1;1;0;"*

7.5 Get Version

Outputs version numbers for the Emulator in addition it outputs the build date/time of the emulation engine

Syntax is as follows:

--getVersion

Output

--versions "<number of components>;<component name 1>;<component version 1>;<component name 2>;<component version 2>;…

Where <number of components> is the number of components for which build numbers are being returned. Then for each component there are two parts:

- Component name – the name of the component e.g.
  - Build – the overall Product Build
  - WebGUI – the Web GUI
  - Ippe – the emulation engine
- Component version/build

Example

```
C:\Users\Lara>python necli.py --host 192.168.202.194 --user admin --password admin --getVersion
--ok "3;Build;1.6.0;WebGui;1.8.5;Ippe;8.0.3 201503241158;"
```

This says we're running version 1.6.0 with a Web GUI version of 1.8.5 and an Emulation engine of 8.0.3 201503241158

8 Methods of Issuing Emulator Commands

The 2 methods of doing this are outlined below:

1. From the Command Line using necli.py
2. Using the Sockets API

8.1 From the Command Line using necli.py

If you want to control Emulator from the command line you can use the provided script necli.py. That implies an installed python language interpreter to successfully run it. The scripts have been tested with both python 2.x and python 3.x.

You must also supply necessary parameters for the user and password you want to use as well as the IP address of the Emulator you want to control and last but not least the task you want to perform.

In generic terms that looks like this:

```
$ python necli.py --host <Emulator IP Address> --user <username> --password <password> --command [--options…]
```

You'll be interested in the result of your commands, so sending the output to a file by adding >filename (and then scanning this file) or piping the output to another program is a good idea.

8.2 The Sockets API

If using the second, direct TCP socket method, a script is not used and instead just the command and options are specified and sent through a socket.

There is a wrinkle though, and that is that the communication must be done using secure sockets (SSL) and this creates a few extra steps. The first thing is to get the remote certificate for the SSL connection. Then the commands can be issued. The two subsections below detail this with python source examples.

8.2.1 Get Remote Certificate

The first thing to do is to retrieve the SSL certificate from the Emulator and store this in a file where it can be used later.

In our sample python script that is done like this:

```
def getRemoteCert(self):
        certData = ssl.get_server_certificate((self.host, self.port),
                                              ssl_version=ssl.PROTOCOL_TLSv1)

        if not len(certData):
            raise IneError('Unable to retrieve certificate from the server')

        try:
            fpath = tempfile.gettempdir() + '/server.crt'
            f = open(fpath, 'w')
            f.write(certData)
            self.cert = f.name
            f.close()
        except:
            raise IneError('Unable to store server certificate to establish
SSL connection')
```

In summary this amounts to:

- Request the remote certificate from the Emulator with IP address self.host using port self.port (7292) using the python library method `ssl.get_server_certificate` with SSL version set to `TLSv1`
- Store the certificate in a temp file with the name `server.crt`

8.2.2 Issue the commands

Having got the certificate (which it is clearly not necessary to do for each command) you can use it to send commands to the Emulator.

The method showing python source examples is:

- Create a socket

```
s = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
```

- Wrap it with the SSL layer:

```
sslsock = ssl.wrap_socket(s, ca_certs=self.cert,
cert_reqs=ssl.CERT_REQUIRED,ssl_version=ssl.PROTOCOL
_TLSv1)
```

- Connect the ssl socket – port 7292

```
sslsock.connect((self.host, self.port))
```

- Send command and options (c)

```
sslsock.sendall(c)
```

- Read response

```
data = sslsock.recv(4096).decode()
```

- Process the response and check for errors (--error)
- Further commands? …repeat from Send command and options

- Finally - Close Socket

```
sslsock.close()
```

9 Troubleshooting

9.1 Unicode Error from Client

Sometimes (particularly on Windows) you may see errors similar to this:

```
C:\Users\Lara>python necli.py --host 192.168.202.194 --user
admin --password admin -getVersion

--error "charmap' codec can't encode character '\u2013' in
position 17: character maps to <undefined>"
```

If you look carefully there is only one – in front of the ***getVersion*** option (even though you're convinced you typed two of them) so this appears to be the problem. You therefore add another – and get exactly the same error.

What has happened here is that perhaps while documenting with MS Word it changed your original --getVersion (i.e. minus minus getVersion) into –getVersion (longdash getVersion) because this looks nicer to read. Longdash is however not an ascii (7 bit or 8 bit) character and has the Unicode value of 2013 – hence the error message. To fix it make sure you're using two proper (short) dashes.